Journal of Discrete Mathematical Sciences & Cryptography
ISSN 0972-0529 (Print), ISSN 2169-0065 (Online)
Vol. 26 (2023), No. 2, pp. 529–559
DOI : 10.1080/09720529.2021.1961900

A new hash function inspired by sponge construction using chaotic maps

Amine Zellagui *
Naima Hadj-Said †
Adda Ali-Pacha §
Laboratory of Coding and Information Security
University of the Sciences and the Technology of Oran
Mohamed Boudiaf
USTO-MB, P. O. Box 505
El M'Naouer Oran 31000
Algeria

Abstract

This paper presents a new secure hash function structure inspired by the sponge construction for hash functions. It is designed to avoid the length of the extension attack, multi-collisions attack, and other vulnerabilities that have been found for many of the hash functions based on the older Merkle Damgård construction. The proposed hash function generates a variable-size hash code. It uses three nonlinear functions that generate chaotic maps: Zaslavsky 2D maps, logistic maps, and Henon maps. Other operations include XOR, rotation, multiplication, and addition. The proposed scheme has a high sensitivity to the initial conditions and message, strong and stable diffusion and confusion capabilities, as well as flexibility in terms of hash size and number of rounds.

Subject Classification: *94A62, 94A60, 68P25, 11K45, 81P94, 37D45.*

Keywords: *Cryptography, Hash, Chaos, SHA-3, Sponge.*

1. Introduction

Hash algorithms play an essential role in cryptography and network security applications. They are mainly used in the context of digital

* *E-mail:* amine.zellagui@univ-usto.dz (Corresponding Author)
† *E-mail:* naima.hadjsaid@univ-usto.dz
§ *E-mail:* a.alipacha@gmail.com

© Taru Publications

signatures with its basic properties. Its aim is to verify the authenticity of the information. Initially hash functions were designed to facilitate database management [49] rather than manipulating data of variable and large size. These data are associated with a fixed size data for which comparisons are quicker to make [35]. This size generally varies between 128 and 512 bits. Most traditional hash algorithms such as MD4 [1], haval-127 [4], RIPEMD [3] and MD5 [2], were successfully attacked by X. Wang in 2004 [5] and SHA-1 [6] by Marc Stevens [7], encouraging researchers to build an efficient hash algorithm.

Recently, the chaos system has become the main concern of researchers which is known to be more sensitive to control parameters, initial conditions and confusion and diffusion properties, uniform data distribution, and random-like behavior [70].

The reproduction of chaos systems seems difficult; this property has encouraged researchers to use chaos in crypto-systems [51, 52, 53, 54, 55], random number generators [57, 58, 59] and hash functions [44, 60,61, 62, 63]. Ahmad et al. [17] proposed a hash function based on six chaotic maps as compression functions. Liu et al. [61] designed a keyed hash function based on hyper chaotic maps with time-varying parameters to process messages. Teh et al. [62] presented a hash function using chaotic systems with fixed point representation. The design use the Merkle-Damgård construction [30] and Feistel structure. Although there has been a countless of new designs throughout the years. The lack of security verification is one of the reasons that hash functions based on chaos maps are not used for real life applications [60, 64]. A hash function should be designed with ease of analysis in consideration, to facilitate cryptanalysis and more secure to resists to known hash functions attacks. But this is not the case of hash functions based on chaos-maps that have been proposed. The most recent chaos-based hash functions are designed with ad hoc structure such as [17, 48, 61], or based on Merkle-Damgård construction [62, 65], which cannot resist to length extension attacks or multi-collision attacks.

This paper introduces a new hash function inspired by sponge construction and based on multiple chaotic maps. It is designed to avoid the length of the extension attack and multi-collisions attack. A one-dimension attractor and two different attractors of two dimensions are used; the logistics map was chosen as the one-dimension attractor and the Henon and Zaslavsky attractor as the two-dimension attractor. Zaslavsky, Henon and logistic maps have been studied and used in cryptography by several researchers [47, 48, 17, 69] to create a pseudo-random number generator (PNRG) in the permutation phase. The sensitivity to control

parameters makes these chaotic maps very useful for the cryptographic hash function which requires a pseudo-randomness,

The reminder of the paper is organized as follows: Section 2 presents the hash construction; Section 3 describes the generality of chaos maps, while Section 4 explains the proposed hash function; Section 5 and 6 present the experimental and theoretical analysis of the proposed hash function; a comparison based on collision resistance and statistical is performed with most recent hash functions in Section 7; finally, Section 8 concludes the paper.

2. Hash construction

A hash function can be composed of two distinct mechanisms. The first mechanism is a transformation that allows to process fixed size data. This transformation is very often a compression function or a permutation. The second mechanism, called a hash construction or domain extension algorithm, is a construction that links the results of the internal transformation to each other. A good domain extension algorithm should ideally preserve the cryptographic properties of the internal function [36, 71].

2.1 *Sponge construction*

The sponge construction is attributed to Bertoni [17]. It is an iterated construct capable of producing arbitrary size outputs. Unlike the Merkle-Damgàrd construction, which is based on a compression function, the sponge construction relies on a fixed-size transformation, i.e. permutation *f*, operating on b-bit words. In practice, all known instances of this construct

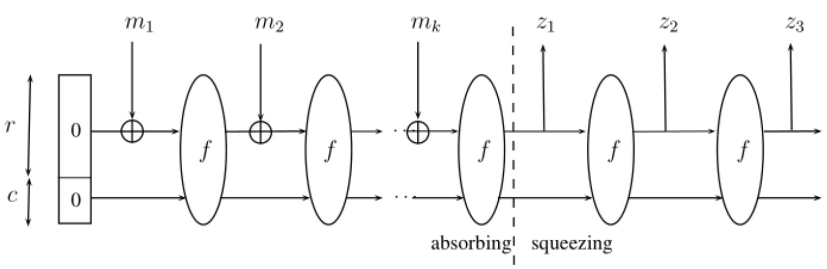


Figure 1

Sponge construction.

are based on a permutation. The procedure takes place in two successive stages (Figure 1):

- The absorbing step.
- The squeezing step

Figure 1 shows the sponge construction, where r is the bit rate and c is the capacity. Both are initially set to zero.

The sponge construction has been used as a domain extension algorithm for several newly introduced hash functions. Among them is the new SHA-3 standard.

3. Generality of chaos maps

The dynamic behavior of nonlinear systems has generated considerable practical interest. Chaos can be generated by a nonlinear dynamic system rather than by a simple structure. A simple recurrence equation can produce a rather complex and rich chaotic dynamics [37].

Chaotic signals have several distinct features, such as: Pseudo-random and unpredictable in the long run, deterministic and high sensitivity to initial conditions and system parameters. [72]

This section introduces the chaotic maps applied in the proposed scheme.

3.1 *Zaslavsky map*

Zaslavsky map is a two-dimensional chaotic map created by Zaslavsky [11]. It is a dynamical system that exhibits chaotic behavior and is function defined as:

$$x_{n+1} = \{x_n(1+\mu y_n\} + \varepsilon v\mu \ .\cos(2\pi x_n)](\text{mod } 1) \quad (1)$$

$$y_{n+1} = e^{-r}(y_n + \varepsilon.\cos(2\pi x_n)) \quad (2)$$

where, ε, v, μ and v are the controlling parameters with $\mu = \frac{1-e^{-r}}{r}$, the point (x_n, y_n) is system variable and x_0, y_0 are the initial values. The Figure 2c shows the Zaslavsky chaotic attractor, while the figure 2a and figure 2b shown the plot of data x and y.

The sequence which generated from Zaslavsky chaotic map has good uniform distribution and can be very useful for cryptography applications.

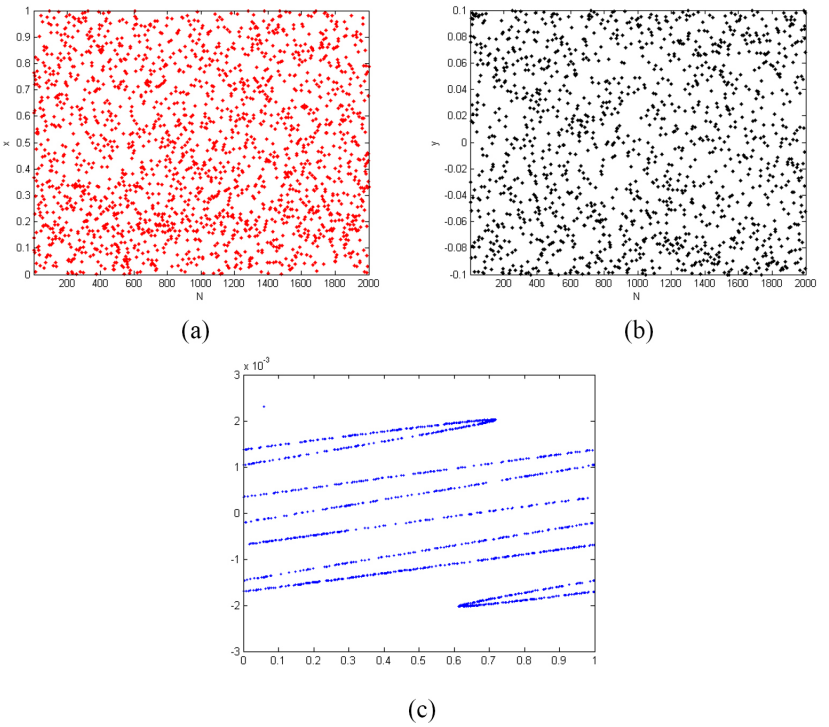


Figure 2

(a) Plot of x, (b) Plot of y, (c) Zaslavsky chaotic attractor.

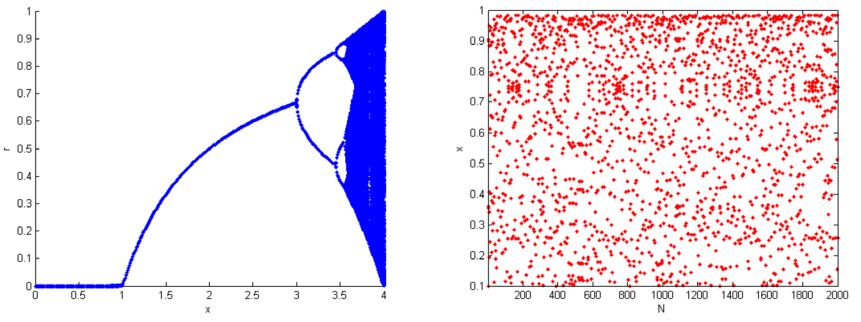
3.2 *Logistic map*

The logistic map is one of the most widely used maps in cryptographic applications. It is a 1D chaotic map familiarized by Robert May in 1976 [12]. The histogram of the generated sequences is not uniform. Its recurrence equation is given by:

$$x_{n+1} = rx_n(1-x_n) \tag{3}$$

where $x_n \in [0,1]$ represents the system state-variable and $r \in [3.6,4]$ represents the system parameters. Figure 3a and Figure 3b shows the bifurcation and the plot of data x of logistic map with parameter *r*.

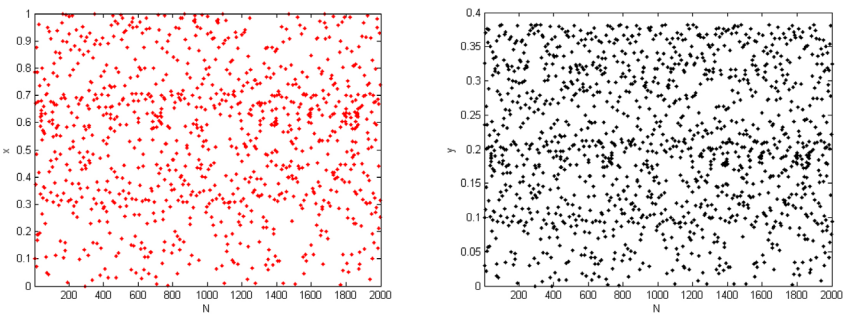
In recent years, several researchers have criticized logistic maps in cryptography applications that are described in [40], but in this work, they are used differently. Section 4 presents their utilization.



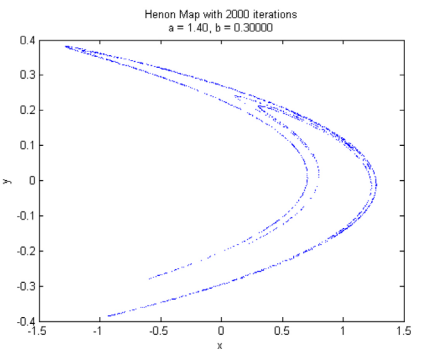
(a) (b)

Figure 3

(a) Bifurcation of 1-D logistic map, (b) Plot of x



(a) (b)



(c)

Figure 4

(a) Plot of x, (b) Plot of y, (c) Henon map attractor.

3.3 *Henon map*

The Henon map is a 2D chaotic map created by Michel Henon [13], it is a discrete-time dynamical system. The Henon map takes a point (x_n, y_n) (Figure 4), and it is mathematically defined as:

$$x_{n+1} = 1 - ax_n^2 + y_n \quad (4)$$

$$y_{n+1} = bx_n \quad (5)$$

where, a and b are control parameters with a > 0, b > 0 and x_n, y_n represents the system state-variable.

4. Scheme of the proposed Hash function

4.1 *Construction*

The proposed hash function is based on a new construction and is inspired by the sponge construction to avoid the length of extension attack, multi-collisions attack and other vulnerabilities of the Merkle-Damgård construction. The aim is to increase the internal state of length *L* of the hash function and to generate the output of length *h*. The absorbing phase takes sequences with length L as input and generates two sequences of bits; the first sequence with length *L* and the second with length *h*.

Figure 5 presents the construction of the proposed hash function inspired by sponge construction, where:

E0 is the initial vector '00....0' of length h (*E0* can be used as a secret key for keyed hash function), *f* is the absorbing phase and g is the chaotic-map functions (Zaslavsky and logistic maps).

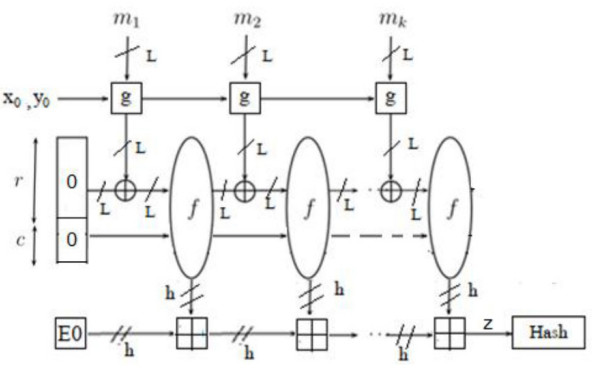


Figure 5

The construction of the proposed hash.

The final hash can be obtained by using the addition function between *E0* and all the output of *f* with length *h*.

4.2 *Hash function*

The proposed hash function is used to compute a message digest of length k= 128,256 or 512 bits for a message or data file with block of length L = 512, 1024 and 2048 bits respectively, provided as input. The following steps were performed to calculate the digest of the message.

A. Padding: One bit "1" was appended to the message, then "0" bits were appended so that the size(in bits) of the padded message becomes congruent to 384 modulo 512 (z bit). Then, a 64-bit was added to the result, where 64-bits represented the size of the message m before the bits were added. Finally, 64-bits were added in the first message, where 64-bits represented the number of blocks after padding (multiple of 512) in binary.

The resulting message (after padding) had a length that is an exact multiple of 512 bits. (Figure 6 shows the padding of the input message *m*)

It is important to note that for a block of length 1024 and 2048 bits, the same steps were followed in blocks of length 512 bits by changing 384 modulo 512 with 768 modulo 1024 and 1536 modulo 2048 bits respectively. Then 128 bits and 256 bits were added at the end and beginning of message for L=1024 and 2048 bits respectively. This process prevents collision between the latest block of different message and produces a protected hash value from the length of extension attack

B. Zaslavsky map: The second step was to transform the message block (in binary) to ASCII (decimal) then to generate new sequence by using Zaslavsky map. To this end, a multiplication between $x_z(n)$, $y_z(n)$, M and inverted M (the first bytes becomes latest and latest bytes become first and so on) was needed at every round. Algorithm 1 describes this process.

Algorithm 1:

- input : $x_z(0)$, $y_z(0)$, ε, v, μ, v
- Output : $x_z(n)$, y_z (n)

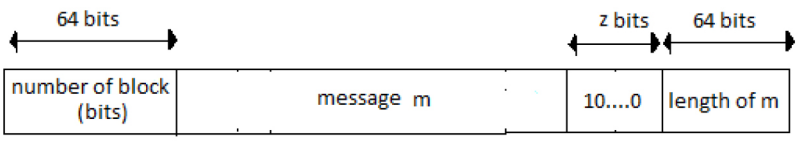


Figure 6

Padding of input message *m*.

- for n = 1, 2, 3, .., N
- $x_z(n+1) = ([\{x_z(n)\ \mathrm{v}(1+\ \mu y_z(\mathrm{n})\} + \varepsilon v\mu\ .\cos(2\pi x_z(n))]\ (\mathrm{mod}\ 1)].\mathrm{M(n)})\ \mathrm{mod}\ 256$
- $y_z(\mathrm{n+1}) = ([e^{-r}(y_z(n) + \varepsilon.\cos(2\pi x_z(n)))].\ \mathrm{M(N-n+1)})\ \mathrm{mod}\ 256$
- end

where, N is L/8 ,M is the message block in decimal (64,128 or 256 bytes) and x_z, y_z is generated by Zaslavsky map. The reason to use M and inverted M (N-n+1), is to obtain high sensitivity to any change in the first and last bit of message M, so the result will be completely changed.

Next, the new sequence M' was calculated as follow:

$$M' = x_z(n)\ XOR\ y_z(n) \tag{6}$$

Figure 7 presents the process of Zaslavsky map.

It is noteworthy that the last value of $x_z(n)$ and y_z (n) will become the initial value in Zaslavsky-map for the next block of message and so on. to make the process more sensitive to any change.

C. Logistic map: The third step represented the transformation of sequence M' into binary code (512, 1024 or 2048 bits) and the permutation between every bit using logistic map, Algorithm 2 illustrate this process.

Algorithm 2:

- input : $x_L(0)$, r
- Output : $x_L(n)$
- for n = 1, 2, 3, .., L

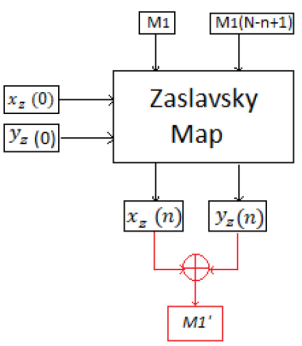


Figure 7

The process of Zaslavsky map.

- $x_L\,(n+1) = (rx_n\,(1-x_L)\,.\,10^7)$ mod (L-n)
- $\mathrm{BLOC_n} = M'\,(x_L\,(n+1))$
- $M'\,(x_L\,(n+1)) = [\,]$;
- end

The aim was to create new matrix named BLOC with length *L*, using *M'* and x_L, where x_L is generated by logistic map. The number of bits in M' decreased in every round until up to zero and increased in BLOC until up to *L*. In addition, mod (L-n) was added by the number of bits in every round, to avoid duplication of the $x_L(n)$ value. Therefore, the last value of $x_L(n)$ became the initial value in logistic-map for the next block of message and so on.

- The fourth step consisted in combining BLOC with the state b using an exclusive OR (XOR), where: r is the bit rate with the same length of BLOC, c is the capacity with length 128, 256 or 512 bits and b = r || c. The result is divided into four equal parts A, B, C, D of length h=160, 320 or 640 bits each.

E. Henon map: The fifth step aimed to calculate the sequences Xi, Yi, Xi2, Yi2 and the variable X' and Y' using Henon map with the sequence A and D (from the four equal parts A, B, C, and D). All the sequences and variable were calculated based on Algorithm 3 and 4.

Algorithm 3

- input : $A_H = 0$, A
- Output : A_H
- for n = 1, 2, 3, .., $\frac{h}{8}$
- $A_H = A_H$ XOR $A(n)$
- end

Algorithm 3 concatenated the sequence A with the variable A_H as newA of length *hlen*, where $hlen = \frac{h}{8} + 1$.

Algorithm 4

- input : $x_H\,(0)$, $y_H(0)$, a, b
- Output : x_n, y_n
- for $n = 1, 2, 3, .., hlen$

- $x_H\,(n+1) = (1 - ax_H^2 + y_H(n)$ newA (n)) mod 256
- $y_H\,(n+1) =$ (b. $x_H\,(n)$ +newA ($hlen - 1$) mod 256
- end

Algorithm 4 set the new sequence x_H and y_H as Xi and Yi respectively with n= {1, 2, ... $hlen - 1$}, then it set x_H as X′ with $n = \{hlen\}$.

The steps to calculate the sequence Xi2, Yi2 and the variable Y′ from D were the same with A, using Algorithm 3 and 4 and replacing A with D.

F. In the sixth step, some operations were performed using addition function with Xi, Yi, B and C and left rotation with X′ and Y′ to obtain new parts A, B, C, Das demonstrated in Algorithm 5.

Algorithm 5

- input : Xi,Yi,B,C,Xi2,Yi2,X′,Y′
- output : A,B,C,D
- for I = 1: r′
- temp1= (ROTL (Xi +B, X′)) mod 2^h
- temp2= (temp1 + Xi2) mod 2^h
- temp3= (ROTL(Yi2 + C,Y′)) mod 2^h
- temp4= (temp3 + Yi) mod 2^h
- D= (C + Yi2) mod 2^h
- C=temp2
- A= (Xi + B) mod 2^h
- B=temp4
- end

where, r′ is the number of round (it can be changed by the user and the fourth parts A, B, C and D will be updated depend on the number of rounds r′). In this step, A, B, C, and D were transformed into decimal value.

G. Squeezing phase: The final step contained the squeezing part, where Z was calculated using the addition function between E0 (initial vector) and the result of A, B, C and D (Figure 8) as follow:

$$Z = (E0 + A+B+C +D) \bmod 2^h \tag{7}$$

where, k is the length of final hash (Hash) and $E0$ is the initial vector ‘00...0’ of length h.

The final hash output with length kis a part of Z, where H’s value is the first k bits of Z.

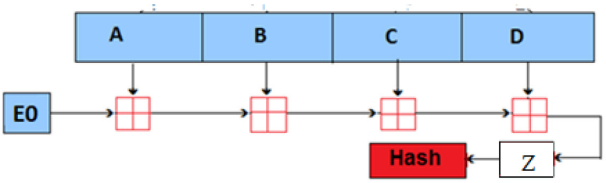


Figure 8

Squeezing function of the proposed scheme (for one block).

It is important to note that if there are more than two block of message, the final hash in block one become the *E0* in block two and the four final part (A,B,C,D) of block one Xored with the first part (A,B,C,D) in block two and so on. to make all message blocks related with each other, these ensure that every bit of the hash is related to all the bits of the original message (Figure 9)

Figure 9 shows the final schematic diagram of the proposed hash function.

5. Experimental analysis

5.1 *Performance*

This subsection presents several tests that were carried out to determine the performance of the proposed scheme. The performance was evaluated against the scan suite that includes hash distribution,

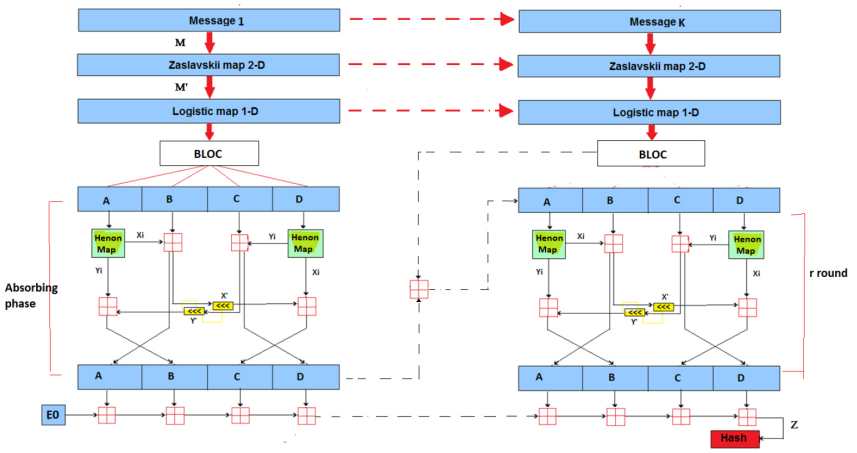


Figure 9

Schematic diagram of proposed scheme.

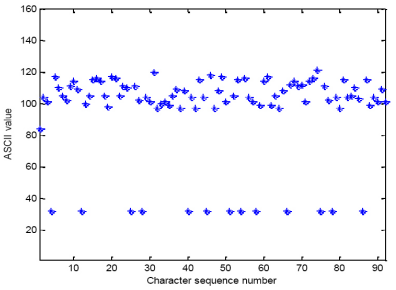


Figure 10a

Random of message characters.

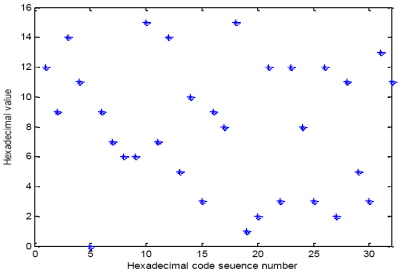


Figure 10b

Hash value with the proposed scheme.

message sensitivity, and flexibility. In addition, a comparison with some recent hash function based on chaos-map was produces. The proposed hash function was implemented in Matlab on Intel Core i5-3200M CPU 2.6 GHz with 4 GB RAM and Windows 7 OS. The values of initial conditions and control parameters were: x0=0.14,y0=0.15, a= 1.4, v= 4 and r=3.9.

5.1.1 Distribution analysis

The distribution analysis of the hash value is important property of hashing scheme; therefore, a message was generated with random characters, then transformed to decimal values plotted in Figure 10a. The hash values of the algorithm are uniformly diffused over the range of hash values (Figure 10b). This ensured that the hash distribution was sufficiently uniform to hide information and provide an effective security measure.

5.1.2 Sensitivity analysis in message and initial conditions

This subsection presents the sensitivity analysis of the proposed hash algorithm with little changes in original message and initial conditions. The simulation for sensitivity was performed as follow:

a. Condition 1: Plaintext is: « The article introduced a new hash function inspired by sponge construction»
b. Condition 2: Replace the first character of the plaintext “T“ by “t“
c. Condition 3: Replace the last character of the plain text “n“ by “N“
d. Condition 4: change x0=0.009 to 0.0091.
e. Condition 5: change y0=0.003 to 0.00301.

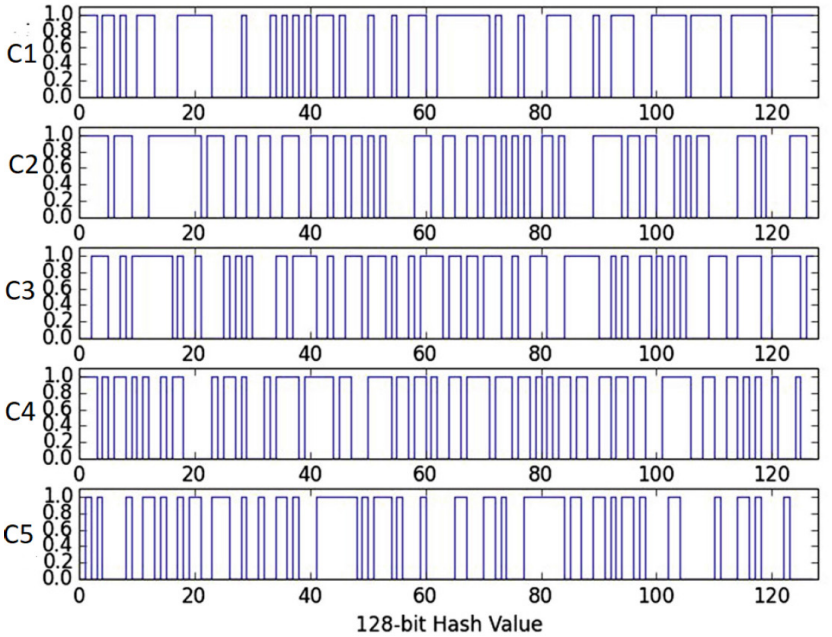


Figure 11

Hash value of the proposed scheme with different conditions.

The corresponding 128-bit hash values in hexadecimal format are the following:

1. Condition1:DD235FD06D8C31FEBC074B5AA83A8AD6
2. Condition 2: 4971194E1172D0312DB1D606A1BC4D97
3. Condition 3: 7B0DE40C93883F5A31971ADA9324D157
4. Condition 4: FD9F2B0B62F854C78EBF7C4B288D2538
5. Condition 5: B772C57411F1AC5DDC56A8A303956BF0

Figure 11 presents all the hash value of the proposed scheme with different conditions.

Based on the simulation results, it can be observed that a small change in message, initial condition and control parameter can change the entire hash value, therefore, the proposed scheme is highly sensitive. Therefore, the proposed hash algorithm is sensitive to any little change in initial parameters.

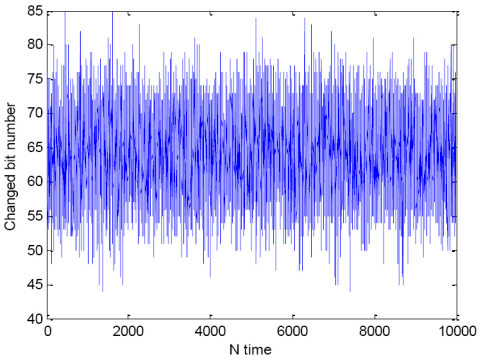


Figure 12

Spread of changed bits number.

5.2 *Security*

5.2.1 Statistical analysis of diffusion and confusion

Due to hash security, in 1949, Shannon developed two features named confusion and diffusion [14] to measure the performance of hashing algorithms and hide the message redundancy. Any slight changes in original message should have 50% changing probability for each bit of hash value. To perform a statistical analysis of confusion and diffusion, first, the hash value of original message must be calculated, then 1 bit of original message changed and the hash value calculated. Next, the two hash values are compared to obtain the number of changed bits. This step will be repeated N times, where N=256, 512, 1024 and 10000. (Figure 12).

The evaluation of confusion and diffusion capabilities is considered by the following measures:

Minimum changed bit number:

$$B_{\min} = \min(\{B_i\}_1^N) \tag{8}$$

Maximum changed bit number:

$$B_{\max} = \max(\{B_i\}_1^N) \tag{9}$$

Mean changed bit number:

$$\overline{B} = \sum_{i=1}^{n} \frac{B_i}{N} \tag{10}$$

Table 1

Statistical result for N = 256, 512, 1024 and 10,000 of the proposed scheme.

Parameter	256	512	1024	10000	Mean
B_{min}	51	51	46	42	47.50
B_{max}	79	79	85	85	82
$\overline{B}$	64.109	63.823	63.807	64.019	63.948
P%	50.085	49.893	50.010	50.015	49.997
ΔB	5.641	5.413	5.634	5.622	5.57
ΔP	4.407	4.224	4.3937	4.392	4.35

Mean changed probability:

$$P = \frac{\overline{B}}{128} \times 100\% \tag{11}$$

Standard variance of the changed bit number:

$$\Delta B = \sqrt{\frac{1}{N-1} \times \sum_{i=1}^{N} (B_i - \overline{B})} \tag{12}$$

Standard variance of probability:

$$\Delta P = \sqrt{\frac{1}{N-1} \times \sum_{i=1}^{N} \left(\frac{B_i}{128} - P \right)} \times 100\% \tag{13}$$

The desired value of $\overline{B}$ should be *L/2.* the desired distribution of hash function should be that little change in original message cause 50% change of hash value. ΔB and ΔP show the stability of diffusion and confusion properties. The hash algorithm is more stable if these two values are close to 0. Based on the results of Table 1, it can be observed that the mean changed bit number $\overline{B}$ and the mean changed probability P of proposed algorithm are nearly the ideal values. All values of ΔB and ΔP are highly small, which signifies that confusion and diffusion capabilities of the proposed hash algorithm are stable and strong.

5.2.2 Resistance to collision attack

Collision resistance is one of properties of cryptographic hash functions; a cryptographic hash function *H* is a collision resistant if it is difficult to find two entries that give the same hash value. However, it is not easy to provide a mathematical proof for the collision resistance of the proposed hash function. The following test was performed to evaluate the

Table 2

Absolute differences of two hash values for N = 512, 2048 and 10000.

	Min	Max	Mean	Mean/n
N=512	673	2195	1434	84.592
N=2048	673	2261	1467	85.647
N=10000	509	2379	1444	85.462

collision resistance as mentioned in [45] and [46]. The collision resistance of the proposed hash function was examined as follows:

First, the hash value of original message was obtained and stored in ASCII code; then, the original message of one-bit modification was applied and a new hash value was obtained. At this point, it is possible to obtain positions where they have the same character by comparing these two-hash value. The absolute is calculated as follows:

$$d = \sum_{i=1}^{N} \left| t(e_i) - t(e_i') \right| \tag{14}$$

where, e_i and e_i' are the ASCII characters of the first and second hash respectively, d is the absolute difference and t is the function that changing the ASCII code into corresponding decimal system value. Table 2 summarizes the results. The last column was obtained by dividing the mean of d by n, where n=l/8 and l is the length of the hash value.

Based on the results of Table 2, it can be observed that the mean per character for the proposed hash function are 85.59, 85.64 and 85.46 for N=512, 2048 and 10000 respectively, and closed to the ideal value of 85.33 [24].

6. Theoretical analysis

6.1 *Flexibility*

The proposed hash function can solve many problems in hash function algorithms including hash value size, padding and resistance to many existing attacks such as those presented above. The proposed scheme can generate hash value of size 128,256 and 512 bits. The number of rounds is not fixed and can be changed by user, to improve the execution time.

6.2 *Resistance to birthday attack*

Birthday attack is a typical attack to break hash function where the attacker needs $2^{l/2}$ items to have a 50% chance of collision [38]. If we have

a hash value of length 64 bits, the attacker only need 2^{32} sattempts, to have 50% chance of collision.

As a result, the hash value length should be increased above 128 bits. The proposed scheme can be generated to 128, 256 and 512 bits. As a result, the proposed scheme is robust against a birthday attack.

6.3 *Resistance to the length extension and multi collisions attacks*

6.3.1 The length extension attack

The most traditional hash functions that are based on Merkle-Damgård construction and other designs cannot resist these attacks, where a message M is divided into n blocks, and H is the hash of M. Another message M' with n+k blocks, where M and M' have the same n block (M1=M1', M2=M2',.., Pad=Mn'), the hash value H(M) is the intermediate hash value after n blocks when computing H(M') (Figure 13) , it can create a problem in practice [31] [34].

$$H(M) = H(M') \text{ after n block} \tag{15}$$

One solution was proposed to avoid these attacks by using padding in the first and last block, where, first padding Pad2 is the length of message in bits, the second Pad1 is the number of blocks in bits (after adding Pad2) and Pad1 || M0 = M1. Its mean that M and M' have not the same n block (M1 ≠ M1', M2 ≠ M2',.., Pad2 ≠ Mn'), and the result H is completely changed after n block.(Figure 14)

$$H(M) \neq H(M') \tag{16}$$

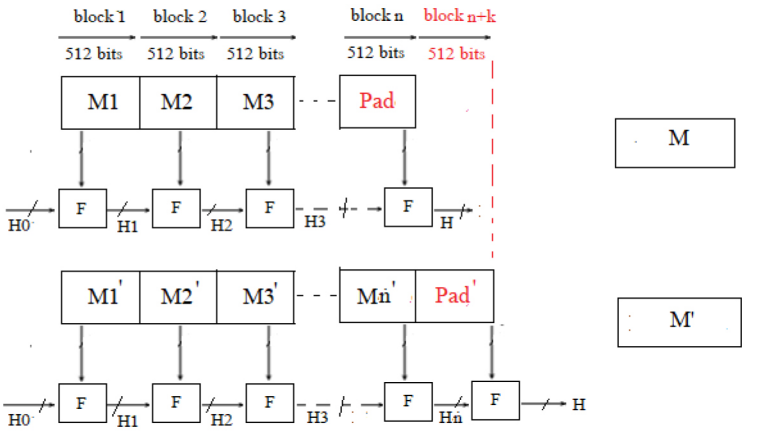


Figure 13

Length extension attack.

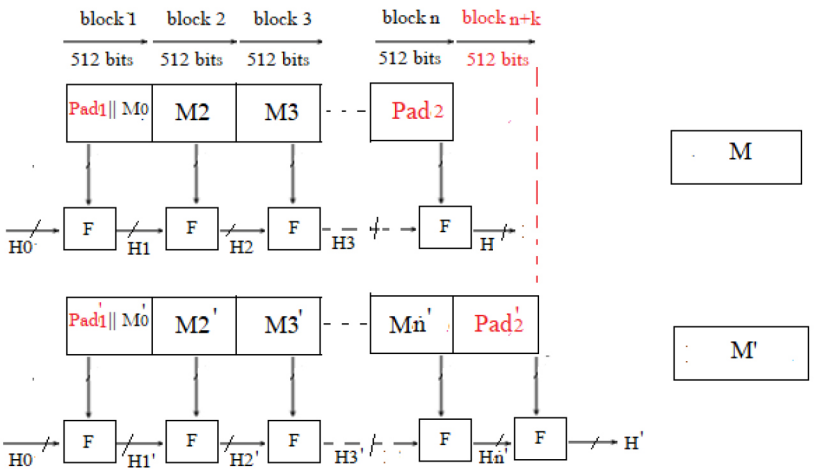


Figure 14

The proposed solution to resist the length extension attack.

6.3.2 Multi collisions attacks

Another problem in Merkle-Damgård construction is the multi-collisions attack. For example, in an iterated hash function, multi-collisions can be constructed more efficiently using internal collisions. First, a pair of messages m0, m0' is searched, which gives an internal collision starting from the *IV*. Then a pair of messages m1, m1'is searched, which gives an internal collision starting from the state x0 after having processed m0. If this operation is repeated k times, k internal collisions are obtained, and 2^k messages can be built by choosing one or the other messages at each step (Figure 15).

The security of the hash function can be reduced to the security of the compression function, but that does not mean that an attack on the compression function gives an attack on the hash function. In particular, if the internal state is sufficiently large compared to the output size n, it is possible to have a hash function that is resistant to multi-collision or pre-image attacks even if it is easy to find collisions or pre-images when we control the chaining variable [41]. The proposed hash function designed to

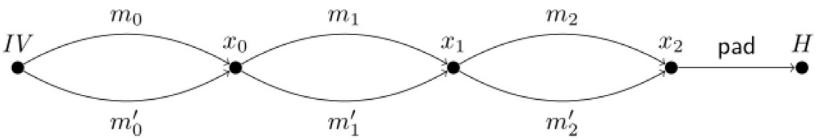


Figure 15

Multi-collisions attack.

avoid the multi-collisions attack. the idea is to increase the internal state L of the hash function using new sequence which generated by the function g (completely different from Mi) and generate the output of length L-bits using the absorbing phase f, after processing all the blocks, the hash output of length h-bits is generated with the compression function. Therefore, with two message Mn and Mn+k with same n block, it is very difficult to have a multi-collision attack. These results demonstrate that one of the proposed schemes is to resist to length extension attack and multi-collision attack.

6.4 *Cryptanalysis*

Hash functions have good cryptographic properties if they are resistant to pre-image and second-pre-image.

Brute force attack is fastest way to compute these attacks; it randomly picks the values of message and tries every value until a collision occurs., The level of effort is proportional to 2^n, where n is hash size. The proposed hash function can be generated 128, 256 or 512 bits, which means that the level of effort is proportional to 2^{128}, 2^{256} or 2^{512} operations, considered very high. In older designs, the entire message is used directly in the compression function f, which will facilitate cryptanalysis attacks, but in the proposed design, the message is passed from function g firstly to generate new sequence where is completely different from M, then, the absorbing phase f takes it as input and generate new one as output with the same size, and finally the compression function generates Z and the final hash.

The proposed chaos-map based hash function use three nonlinear chaotic-map. Zaslavsky maps used as a pseudo-random-number generator to change data before the absorbing phase step. Logistic map used as a permutation function between every bit, these steps are one way, and it's hard to back to original message M. Henon map is used as a transformation function in the absorbing phase f, to make process more sensitive to any little change, this makes it very difficult to have a collision in absorbing

6.5 *Meet-in-the-middle attack (MITM)*

In 2008, Aoki and Sasaki proposed the meet-in-the-middle attack (MITM) [42]; it uses the message schedule that consists of permutations of message words. It is successfully applied on SHA-1 [43], where the authors have used the linear combination of message words.

The proposed scheme uses three nonlinear functions that generate chaotic maps: Zaslavsky 2D maps, logistic maps, and Henon maps. In

every round, the values x^n, y^n, M' and *BLOC* are derived from message blocks. In addition, they are no fixed words or values which means that the function is dependent on the message. Therefore, this attack cannot be applied to the proposed hash function.

7. Comparative analysis

This subsection presents the comparison of the proposed scheme with other recent hash functions based on chaos-maps, SHA-2 and SHA-3. This comparative analysis includes the statistical analysis of diffusion and confusion, collision resistance and speed analysis.

7.1 *Comparison of statistical performance*

The proposed hash function was compared with some recent hash functions based on chaos-maps, Table 3 and 4 summarize the result of the statistical performance comparison and it can be observed that our hash function has a superior statistical analysis than some hash function such as: Kanso's ,Lin's, Ahmad's, Deng's , Li's, Guo's ,SHA-2, Xiao's , Zhang's, Yantao, Teh and SHA-3.

Table 3

Statistical performance comparison (N = 2048 with 128-bit hash)

Algorithms	$\overline{B}$	P%	ΔB	ΔP%
Kanso's [15]	63.93	49.96	5.69	4.44
Lin's [16]	63.84	49.87	5.76	4.50
Ahmad's [17]	63.87	49.90	5.85	4.36
Deng's [18]	63.48	49.89	5.88	4.59
Li's [19]	63.89	49.91	5.64	4.41
Xiao's [20]	64.09	50.01	5.48	4.28
Guo's [22]	63.40	49.53	7.15	6.35
Zhang's [23]	64.43	49.96	5.57	4.51
Teh [60]	64.02	50.01	5.64	4.40
Yantao [63]	63.94	49.96	5.61	4.38
Ren [67] CM	63.99	49.99	5.68	4.46
Ren [67] DA	63.89	49.91	5.66	4.42
Ren [67] SA	63.49	49.95	5.67	4.43
Mihaela [68]	64.11	50.08	5.59	4.37
Proposed	64,12	50.05	5.16	4.22

Table 4

Statistical performance comparison(N = 2048 with 256-bit hash)

Algorithms	$\overline{B}$	P%	ΔB	ΔP%
SHA-2	127.85	50.01	7.90	3.25
Hongjun [61]	128.13	50.05	7.94	3.10
Yijun [66]	128.01	50.00	8.17	3.19
Mihaela [68]	128.98	50.38	8.08	3.15
SHA-3	128.56	50.04	8.12	3.18
Proposed	128.66	50.06	7.27	3.12

7.2 *Collision resistance*

Tables 5 and 6 summarize the results of the collision resistance capability comparison of the proposed hash function. The value obtained is very close to the ideal value (85.33) which means that the suggestion is very resistant to collision attack.

Table 5

Absolute distance between two hash values (N = 2048 with 128-bit hash)

Algorithms	Minimum	Maximum	Mean	Mean/char
Kanso's [15]	731	2230	1368	85.50
Ahmad's [17]	537	2399	1364.7	85.29
Deng's [18]	583	2206	1399.8	87.49
Li's [19]	577	2089	1335	83.41
Xiao's [20]	573	2224	1401.1	87.56
Yantao [63]	688	2019	1350	84.38
Teh [60]	584	2321	1366	N/A
Fan's [21]	872	2524	1675	83.75
Mihaela [68]	537	2386	1367	N/A
Proposed	673	2261	1364.8	85.30

Table 6

Absolute distance between two Hash values (N = 2048 and 256-bit hash)

Algorithms	Minimum	Maximum	Mean	Mean/char
SHA-2	1619	3757	2688	84
Yijun [66]	N/A	N/A	N/A	85.334
Hongjun [61]	1883	4051	2887	N/A
SHA-3	1628	3839	2733.5	85.42
Proposed	1627,36	3844	2735.68	85.49

7.3 *Speed analysis*

The proposed hash function was implemented in Matlab 2013. Figure 15 outlines the trade-off between the number of rounds and execution time. It can be observed that whenever the number of rounds increases, the execution time is higher.

In addition, a speed comparison with some popular hash functions is provided in Table 7. The speed test results show that the proposed scheme is better than those proposed by SHA-1, Chenaghlu's, SHA-2 and SHA-3.

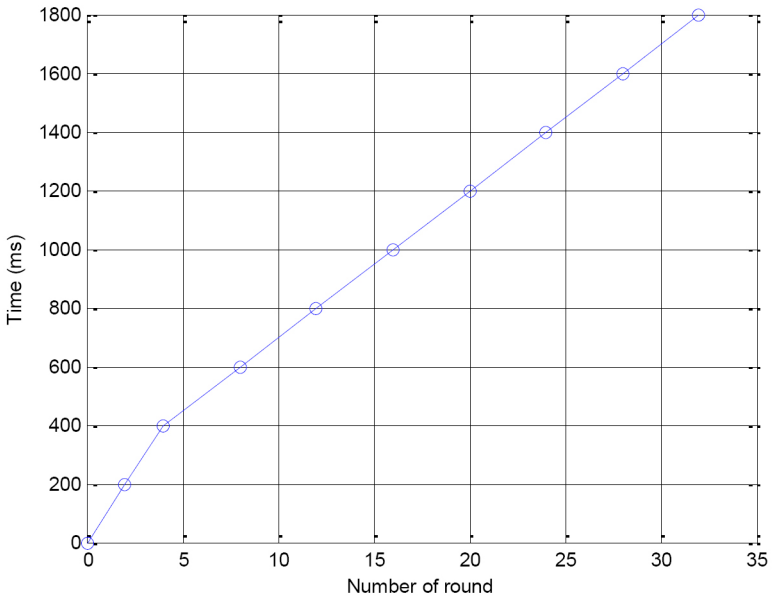


Figure 16

Time performance.

Table 7

Speed performance comparison.

Algorithms	Platform	Speed in Mbps
SHA-1	Intel core i5-3200M CPU @ 2.6 GHzwith 4 GB RAM	600.9
Chenaghlu's [39]	Intel corei3 CPU @ 4 GB RAM on Ubuntu	645.47
Hongjun [61]	Pentium 4 CPU @ 2.50 GHz 2G RAM Ubuntu	132.0
SHA-2	Intel core i5-3200M CPU @ 2.6 GHzwith 4 GB RAM	580.23
SHA-3	Intel core i5-3200M CPU @ 2.6 GHzwith 4 GB RAM	719.56
Proposed	Intel core i5-3200M CPU @ 2.6 GHzwith 4 GB RAM	722.73

7.4 *Discussion*

The proposed hash function is inspired by the sponge construction and based on the results it has a better performance than other hash functions including SHA-2 (based on Merkle-Damgård construction). The proposed hash construction can avoid attacks such as length extension attacks and multi-collision attacks.

In addition, this hash function uses three nonlinear functions that generate chaotic maps including multiplication, addition, permutation to obtain high complexity, with the possibility of reducing the number of rounds to improve the execution time. Thus, it is difficult to derive the original message from the hash.

The state of the zaslavsky map is directly related to every message bit in message blocks. Then, the state of the logistic map is directly related to zaslavsky map, Therefore, the state of henon map is related with the sequence which generated by those chaotic-maps. these ensure that every bit of the hash is related to all the bits of the original message.

Since the chaotic system operates on a real number field, the cryptanalysis of chaos-based ciphers is more complex than the traditional ones. The speed test results show that the proposed hash function is faster than SHA-2 and SHA-3. Furthermore, it can be easily implemented and used for digital signatures, data management and passwords hashing.

8. Conclusion

The article introduced a new hash function inspired by sponge construction. It aims to avoid the length extension attacks, multi-collision attacks and other vulnerabilities of Merkle Damgård construction. The function is also based on multiple chaotic maps using Zaslavsky map 2D, logistic map 1D and Henon map 2D.

The proposed scheme, given the high sensibility of any little change to the original message, the large statistical diffusion and confusion performance, the high collision resistance, can be considered as a key hash function using the initial value x_0 , y_0 and the initial vector E0 as a key. The number of rounds is not fixed and can be changed by the user to improve the execution time**.** The proposed scheme aims to facilitate the management of stored databases and passwords, easy implementation and fast files processing.

References

[1] Rivest R., 1992, "The MD4 Message-Digest Algorithm," RFC 1320, MIT LCS and RSA Data Securit y, Inc

[2] Rivest R., 1992, "The MD5 Message-Digest Algorithm," RFC 1321, MIT LCS and RSA Data Securit y, Inc

[3] Dobbertin, H., Bosselaers, A., & Preneel, B. (1996, February). RIPEMD-160: A strengthened version of RIPEMD. In International Workshop on Fast Software Encryption (pp. 71-82). Springer, Berlin, Heidelberg.

[4] Zheng, Y., Pieprzyk, J., Seberry, J.: HAVAL - A One-Way Hashing Algorithm with Variable Length of Output. In: ASIACRYPT 1992. LNCS, pp. 83–104. Springer, Heidelberg (1992)

[5] Wang, X., Feng, D., Lai, X., & Yu, H. (2004). Collisions for hash functions MD4, MD5, HAVAL-128 and RIPEMD.Cryptologye Print Archive, report 2004/199

[6] Eastlake 3rd, D., & Jones, P. (2001). RFC3174: Us secure hash algorithm 1 (SHA1). Internet RFCs,

[7] Stevens, M., Bursztein, E., Karpman, P., Albertini, A., & Markov, Y. (2017, August). The first collision for full SHA-1. In Annual International Cryptology Conference (pp. 570-596). Springer, Cham.

[8] Eastlake, D., & Hansen, T. (2006). US secure hash algorithms (SHA and HMAC-SHA).RFC 4634 (Informational), Internet Engineering Task Force, Jul. 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4634.txt

[9] Merkle, R. C. (1989, August). A certified digital signature. In Conference on the Theory and Application of Cryptology (pp. 218-238). Springer, New York, NY.

[10] Damgård, I. B. (1989, August). A design principle for hash functions. In Conference on the Theory and Application of Cryptology (pp. 416-427). Springer, New York, NY.

[11] G.M. Zaslavsky (1978). "The Simplest case of a strange attractor". Phys. Lett. A. 69 (3): 145–147. Bibcode: 1978PhLA...69. 145Z. doi:10.1016/0375-9601(78)90195-0.

[12] May, Robert M. (1972-08-18). "Will a Large Complex System be Stable?". Nature. 238 (5364): 413–414. Bibcode:1972Natur. 238..413M. doi:10.1038/238413a0.

[13] M. Hénon (1976). "A two-dimensional mapping with a strange attractor". *Communications in Mathematical Physics.* 50 (1): 69–77. doi:10.1007/BF01608556.

[14] Shannon, C. E. (1949). Communication theory of secrecy systems. *Bell Systems Technical Journal,* 28, 656–715

[15] Kanso, A., Yahyaoui, H., & Almulla, M. (2012). Keyed hash function based on a chaotic map. Information Sciences, 186(1), 249-264.

[16] Lin, Z., Guyeux, C., Yu, S., Wang, Q., & Cai, S. (2019). On the use of chaotic iterations to design keyed hash function. *Cluster Computing,* 22(1), 905-919.

[17] Ahmad, M., Khurana, S., Singh, S., & AlSharari, H. D. (2017). A simple secure hash function scheme using multiple chaotic maps. 3D Research, 8(2), 13.

[18] Deng, S. J., Li, Y. T., & Xiao, D. (2009). Analysis and improvement of a chaos-based hash function construction. *Communications in Nonlinear Science and Numerical Simulation,* 15(5), 1338–1347

[19] Li, Y., Xiao, D., & Deng, S. (2012). Keyed hash function based on a dynamic lookup table of functions. *Information Sciences*, 214, 56–75

[20] Xiao, D., Shih, F. Y., & Liao, X. F. (2010). A chaos-based hash function with both modification detection and localization capabilities.

Communications in Nonlinear Science and Numerical Simulation, 15(9), 2254–2261

[21] Fan, Z., Xiao-Jian, T., Xue-Yan, L., & Bin, W. (2008). Hash function based on the generalized Henon map. *Chinese Physics B*, 17(5), 1685.

[22] Guo, W., Wang, X., He, D., & Cao, Y. (2009). Cryptanalysis on a parallel keyed hash function based on chaotic maps.*Physics Letters A*, 373(36), 3201–3206.

[23] Zhang, H.,Wang,X.,Li,Z.,&Liu,D.(2005).On eway hash function construction based on spatio temporal chaos. *Acta PhysicaSinica*, 54, 4006–4011.

[24] Li, Y., Ge, G., & Xia, D. (2016). Chaotic hash function based on the dynamic S-Box with variable parameters. *Nonlinear Dynamics*, 84(4), 2387–2402

[25] Sklavos, N., & Kitsos, P. (2010, July). BLAKE HASH function family on FPGA: From the fastest to the smallest. In 2010 IEEE Computer Society Annual Symposium on VLSI (pp. 139-142). IEEE.

[26] IDEGUCHI, Kota, TISCHHAUSER, Elmar, et PRENEEL, Bart. Improved collision attacks on the reduced-round hash function. In: International Conference on Information Security. Springer, Berlin, Heidelberg, 2010. p. 1-16.

[27] Wu, H. (2011). The hash function JH. Submission to NIST (round 3), 6. https://perso.uclouvain.be/fstandae/source_codes/hash_atmel/specs/jh.pdf

[28] Bertoni, G., Daemen, J., Peeters, M., & Van Assche, G. (2009). Keccak sponge function family main document. Submission to NIST (Round 2), 3(30), 320-337. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.419.2140&rep=rep1&type=pdf

[29] Ferguson, N., Lucks, S., Schneier, B., Whiting, D., Bellare, M., Kohno, T., ... & Walker, J. (2010). The Skein Hash Function Family. Submission to NIST (Round 3). https://www.schneier.com/wp-content/uploads/2008/10/skein.pdf

[30] I. B. Damgård, "A design principle for hash functions," in Proc. Crypto'89, Santa Barbara, CA, Aug. 1989, pp. 416–427.

[31] Denton, B., & Adhami, R. (2011). Modern hash function construction. In Proceedings of the International Conference on Security and Management (SAM) (p. 1). The Steering Committee of The World

Congress in Computer Science, *Computer Engineering and Applied Computing* (WorldComp).

[32] A. Joux, "Multicollisions in Iterated Hash Functions: Application to Cascaded Constructions," Crypto'04, 2004, LNCS, vol. 3152, pp. 306-316

[33] Bertoni, G., Daemen, J., Peeters, M., & Van Assche, G. (2008, April). On the indifferentiability of the sponge construction. In Annual International Conference on the Theory and Applications of Cryptographic Techniques (pp. 181-197). Springer, Berlin, Heidelberg.

[34] Niels Ferguson, Bruce Schneier, and Tadayoshi Kohno,Cryptographic Engineering: Design Principles and Practical Applications.: Wiley Publishing, Inc., 2010.

[35] Chi, L., & Zhu, X. (2017). Hashing techniques: A survey and taxonomy. ACM Computing Surveys (CSUR), 50(1), 1-36.

[36] Andreeva, E., Mennink, B., & Preneel, B. (2010). Security properties of domain extenders for cryptographic hash functions. J*ournal of Information Processing Systems*, 6(4), 453-480.

[37] Kocarev, L. (2001). Chaos-based cryptography: a brief overview. *IEEE Circuits and Systems Magazine*, 1(3), 6-21.

[38] Preneel, B., Govaerts, R., &Vandewalle, J. (1994). Hash functions based on block ciphers A synthetic approach. *Lecture Notes in Computer Science*, 773, 368– 378.

[39] Chenaghlu, M. A., Jamali, S., &Khasmakhi, N. N. (2016). A novel keyed parallel hashing scheme based on a new chaotic system. Chaos, Solitons & Fractals, 87, 216–225.

[40] Arroyo, D., Alvarez, G., & Fernandez, V. (2008). On the inadequacy of the logistic map for cryptographic applications. arXiv preprint arXiv:0805.4355.

[41] Zellagui, A., Hadj-Said, N., Ali-Pacha, A. (2019). Comparative Study Between Merkle-Damgård And Other Alternative Hashes Construction, Second conference on informatics and Applied Mathematics IAM'2019, 180, 30–34, GUELMA, ALGERIA

[42] Aoki, K., and Y. Sasaki. 2008. Preimage attacks on One-Block MD4, 63 steps MD5 and more. SAC, LNCS, vol. 5381. New York: Springer,

[43] Aoki, K., and Y. Sasaki. 2009. Meet-in-the-Middle preimage attacks against reduced SHA-0 and SHA-1. Advances in Cryptology - CRYPTO 2009, vol. 5677, 70–89. LNCS. New York: Springer.

[44] Zellagui, A., Hadj-Said, N., & Ali-Pacha, A. (2020). Secure MD4 Hash Function Using Henon. *Malaysian Journal of Computing and Applied Mathematics*, 3(2), 73-80.

[45] Wong, K. W. (2003). A combined chaotic cryptographic and hashing scheme. *Physics letters A*, 307(5-6), 292-298.

[46] Xiao, D., Liao, X., & Deng, S. (2008). Parallel keyed hash function construction based on chaotic maps. *Physics Letters A*, 372(26), 4682-4688.

[47] Chirakkarottu, S., & Mathew, S. (2020). A novel encryption method for medical images using 2D Zaslavski map and DNA cryptography. SN Applied Sciences, 2(1), 1-10. DOI: 10.1007/s42452-019-1685-8

[48] Moysis, L., Petavratzis, E., Volos, C., Nistazakis, H., & Stouboulos, I. (2020). A chaotic path planning generator based on logistic map and modulo tactics. *Robotics and Autonomous Systems*, vol. 124, DOI: 10.1016/j.robot.2019.103377.

[49] Saez, Y., Estebanez, C., Quintana, D., & Isasi, P. (2019). Evolutionary hash functions for specific domains. *Applied Soft Computing*, 78, 58-69.

[50] Guo, J., Liao, G., Liu, G., Liu, M., Qiao, K., & Song, L. (2020). Practical collision attacks against round-reduced SHA-3. *Journal of Cryptology*, 33(1), 228-270.

[51] Al-Khedhairi, A., Elsonbaty, A., Elsadany, A. A., & Hagras, E. A. (2020). Hybrid Cryptosystem Based on Pseudo Chaos of Novel Fractional Order Map and Elliptic Curves. IEEE Access, 8, 57733-57748.

[52] Farah, M. B., Guesmi, R., Kachouri, A., & Samet, M. (2020). A new design of cryptosystem based on S-box and chaotic permutation. Multimedia Tools and Applications, 1-22.

[53] Yu, C., Li, X., Xu, S., & Li, J. (2020). Computer generated hologram-based image cryptosystem with multiple chaotic systems. *Wireless Networks*, 1-15.

[54] Broumandnia, A. (2019). The 3D modular chaotic map to digital color image encryption. *Future Generation Computer Systems*, 99, 489-499.

[55] Li, Y. C., Tang, D. Y., & Ye, R. S. (2019, June). A Novel Color Image Encryption Scheme based on Hybrid Chaotic Maps. In 2019 14th IEEE Conference on Industrial Electronics and Applications (ICIEA) (pp. 1431-1436). IEEE.

[56] Ayubi, P., Setayeshi, S., & Rahmani, A. M. (2020). Deterministic chaos game: A new fractal based pseudo-random number generator and its cryptographic application. *Journal of Information Security and Applications*, 52, 102472.

[57] Tutueva, A. V., Nepomuceno, E. G., Karimov, A. I., Andreev, V. S., & Butusov, D. N. (2020). Adaptive chaotic maps and their application to pseudo-random numbers generation. Chaos, Solitons & Fractals, 133, 109615.

[58] Datcu, O., Macovei, C., & Hobincu, R. (2020). Chaos based cryptographic pseudo-random number generator template with dynamic state change. *Applied Sciences*, 10(2), 451.

[59] Ozkaynak, F. (2020). A novel random number generator based on fractional order chaotic Chua system. *Elektronika ir Elektrotechnika*, 26(1), 52-57.

[60] Teh, J. S., Alawida, M., & Ho, J. J. (2020). Unkeyed hash function based on chaotic sponge construction and fixed-point arithmetic. Nonlinear Dynamics, 100(1), 713-729.

[61] Liu, H., Kadir, A., & Liu, J. (2019). Keyed hash function using hyper chaotic system with time-varying parameters perturbation. *IEEE Access*, 7, 37211-37219.

[62] Teh, J. S., Tan, K., & Alawida, M. (2019). A chaos-based keyed hash function based on fixed point representation. *Cluster Computing*, 22(2), 649-660.

[63] Li, Y., & Ge, G. (2019). Cryptographic and parallel hash function based on cross coupled map lattices suitable for multimedia communication security. *Multimedia Tools and Applications*, 78(13), 17973-17994.

[64] Alvarez, G., Amigó, J. M., Arroyo, D., & Li, S. (2011). Lessons learnt from the cryptanalysis of chaos-based ciphers. In Chaos-Based Cryptography (pp. 257-295). Springer, Berlin, Heidelberg.

[65] Abdoun, N., El Assad, S., Deforges, O., Assaf, R., & Khalil, M. (2020). Design and security analysis of two robust keyed hash functions based on chaotic neural networks. *Journal of Ambient Intelligence and Humanized Computing*, 11(5), 2137-2161.

[66] Yang, Y., Chen, F., Chen, J., Zhang, Y., & Yung, K. L. (2019). A secure hash function based on feedback iterative structure. *Enterprise Information Systems*, 13(3), 281-302.

[67] Ren, H. P., Zhao, C. F., & Grebogi, C. (2020). One-way hash function based on delay-induced hyperchaos. *International Journal of Bifurcation and Chaos*, 30(02), 2050020.

[68] Todorova, Mihaela, BorislavStoyanov, Krzysztof Szczypiorski, and KrasimirKordov. "SHAH: hash function based on irregularly decimated chaotic map." arXiv preprint arXiv:1808.01956 (2018).

[69] Balaska, N., Ahmida, Z., Belmeguenai, A., & Boumerdassi, S. (2020). Image encryption using a combination of Grain-128a algorithm and Zaslavsky chaotic map. *IET Image Processing*, 14(6), 1120-1131.

[70] A. Shakiba (2019) Security analysis for chaotic maps-based mutual authentication and key agreement using smart cards for wireless networks, *Journal of Information and Optimization Sciences*, 40:3, 725-750, DOI: 10.1080/02522667.2018.1470752.

[71] Hamid Mirvaziri (2020) Evolutionary double-pipe prefix-free hash function, *Journal of Information and Optimization Sciences*, 41:5, 1175-1191, DOI: 10.1080/02522667.2019.1662581.

[72] Mohammad Ali Khan, Himadri Pai Mazumdar & Syeda Darakhshan Jabeen (2020) Anti-synchronization phenomenon of discrete chaotic maps using linear transformations, *Journal of Information and Optimization Sciences*, 41:8, 1757-1769, DOI: 10.1080/02522667.2017.1321766

Received January, 2021

Revised May, 2021